Controller synthesis for application specific digital signal processors

Öwall, Viktor; Torkelson, Mats

Published in:
[Host publication title missing]

DOI:
10.1109/ASIC.1991.242899

1991

CONTROLLER SYNTHESIS FOR APPLICATION SPECIFIC DIGITAL SIGNAL PROCESSORS

Viktor Öwall and Mats Torkelson

Department of Applied Electronics
Lund University, Sweden

Abstract

A controller synthesizer, that is part of a design system by which algorithms unsuitable for standard processors can be implemented, is presented. A hierarchical controller architecture suitable for frame-based and multi-sample-rate algorithms is synthesized. Synthesis of a controller is based on micro instructions, specific for each architecture, and assumes no use of predefined functional blocks. The designer can affect complexity and partitioning of the controller by changing the micro program. Processors for speech scrambling and digital adjustment of quadrature modulators have been designed and fabricated.

1 Introduction

Application Specific Digital Signal Processors (ASDSPs) require a complex set of signals to control the data flow through the processor. Various ASDSP architectures require different sets of control signals and different complexity of the controller. Thus, it is important to have a flexible tool for controller synthesis. The presented Control Unit Synthesizer (CUS) is aimed at digital radio communication applications where frame-based and multi-sample-rate algorithms are frequently used. Therefore, our CUS synthesizes controllers which have a hierarchical architecture suitable for these kinds of algorithms [1].

The CUS synthesizes a controller from a micro program and is tightly coupled to a Data Path Compiler (DPC) [2]. The DPC and the CUS put no restrictions on the processor architecture which gives the designer the flexibility to develop an architecture suitable for the algorithm. Algorithms with conditional statements, subroutine calls, conditional subroutine calls, loops, etc. can be implemented with the CUS.

In order to simulate and debug the micro program a Register Transfer Level (RTL) simulator has been developed. The DPC and the CUS together with the RTL simulator make it possible to design complex ASDSPs in a short time.

2 System Overview

Because of the tailored architecture of an ASDSP the CUS has to work closely with the DPC. The DPC generates data path modules from structural descriptions, additionally the DPC generates a behavioral description of the processor. The behavioral description consists of all micro instructions available for the processor, status signals, and default levels to control signals.

The algorithm is described in a micro program with the micro instructions defined by the DPC, memories (ROMs and RAMs) can be declared for usage in the micro program. Subroutines, case statements, and variable passing are used in a way similar to high level programming to make scheduling and simulation easier. A C program which performs an RTL simulation of the micro code can be generated from the micro program. The RTL simulation can be performed both in floating point representation and on bit level and allows the designer to debug the micro code without generation, extraction, and simulation of the chip.

The CUS synthesizes a complete hierarchical controller and specifies its interconnections to data paths and I/O-units. Memory modules, with a supporting Address Processing Unit (APU), are generated by the CUS if such are declared in the input specification. Partitioning and complexity of the controller is dependent on the structure of the micro program. Therefore, the designer can try various strategies in partitioning of micro code and memories, and complexity of APUs, to find a good solution.

The generated ASDSP is finally extracted and simulated at transistor level before fabrication. The tools have been modified to enable the use of different cell libraries and to produce different output formats.

3 Controller Architecture

The controller architecture contains one micro code level and one or several sequencing levels. Each level controls the next lower level, the data path for the micro code, and is controlled by the next higher level, an external signal for the highest level. The number of sequencing levels is decided by the partitioning of the micro code.

The micro code level is the lowest hierarchical level and consists of a micro code ROM, a program counter, and a pipeline register, figure 1.

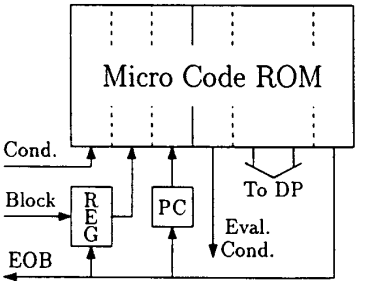

Figure 1: The micro code level.

91TH0379-8/91/0000-P13-5.1$01.00

The micro code ROM generates control signals which control the data flow through the data paths and control signals to the APU. The code is partitioned into blocks of micro instructions of arbitrary length (subroutines). Micro instructions in a block are executed in sequential order, controlled by a program counter, where each time slot is one clock cycle. The sequencing of blocks is controlled by the next higher level ROM. Pipeline registers are implemented between all levels in the controller to avoid clock cycle overhead on out-of-block transitions. At the end of each block an End Of Block (EOB) signal is set that resets the program counter and loads a new block address into the pipeline register. This EOB signal also increments the program counter on the next higher level.

It is possible to select different micro operations or memory locations with a case statement. Case statements use Boolean conditions evaluated from status signals in a Decision Finite State Machine (DFSM). Status signals can be both data path signals, and external signals from I/O-units. The evaluation of conditions is controlled from the micro code ROM. A condition should not be used before it has been evaluated. The DFSM can be implemented in one or many PLAs depending on speed requirements or floorplan considerations, partitioning of the DFSM is controlled by the designer in the micro program.

The next hierarchical levels describe the sequencing of blocks. Each level is divided into blocks and each block into time slots as in the micro code ROM. In the sequencing levels, however, a time slot is not one clock cycle but one block in the next underlying level. Thus a time slot in higher levels is not a fixed number of clock cycles but of arbitrary length decided by the designer. All sequencing levels in the controller are implemented in the same way and is controlled by a counter and by the next higher level, the same way as for the micro code ROM.

The counter sequences through the block addresses for the next lower level. The counter is incremented when an EOB signal is received from the next lower level and reset with the EOB signal from the same level. Loops are realized by disabling the incrementing of the level counter until the required condition in the DFSM is fulfilled. An example with micro code and two sequencing levels is shown in figure 2.

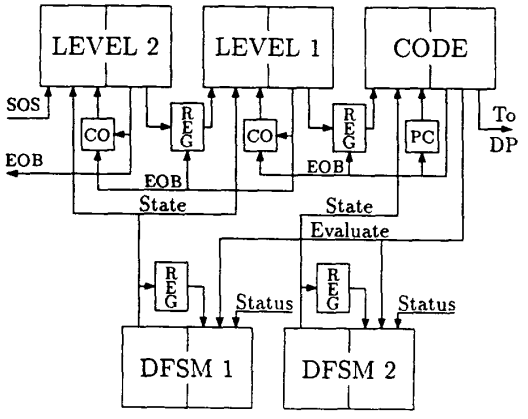

Figure 2: Architecture for a three level controller.

The highest level is controlled by an external Start of Sequence (SOS) signal. At the end of a program sequence the controller will examine the SOS signal. If SOS is set the micro code will be executed once more, otherwise the controller will send default signals to the data path and wait for SOS to be set.

The DFSM is connected to all levels in the controller architecture. Thus, it is possible to use the case statement for making decisions on every level in the hierarchy. On the micro code level case statements are used to choose different sets of micro operations and on higher levels to choose what block address to send to the lower level. Synchronization between different modules in a processor is taken care of by the DFSM.

The controller is partitioned into small modules in order to make the controller faster and make it easier to get a dense floorplan. To avoid one large micro code ROM it can be partitioned into smaller and faster modules to be placed close to the controlled module.

The described controller architecture is best suited for micro code with non nested case and while statements. However replacing the counters with feedback registers results in a finite state machine implementation, more suitable for nested programming [3, 4]. Such an architecture, with kept hierarchy, is currently under development, figure 3.

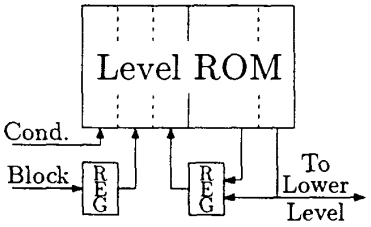

Figure 3: Finite state machine implementation.

Communication between processors is important in large systems. The described system supports both communication between processors on the same chip and communication with external processors. Co-processors can be synchronized to each other using the DFSM and to a host processor using the SOS signal. Future work will be to investigate implementation of parallel processors and parallel controllers and how to synchronize these, figure 4.

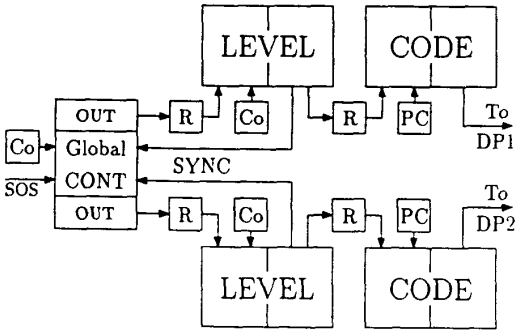

Figure 4: Parallel controllers

4 Memories and Address Processing Unit

Memories, RAMs and ROMs, with an Address Processing Unit (APU) can be generated optionally. The CUS generates description files of declared memories to a memory generator and routing descriptions for the DPC. Output of the memories can be connected to any bus at any data path. If large memories are needed in a design the CUS is prepared for handling external memories. Address bus, read, and write signals are then connected to I/O-units.

The address ROM is separated from the micro code ROM and can be controlled from more than one of the hierarchical levels. Thus, it is possible to execute the same micro code several times with different memory locations by passing variables from higher levels. Otherwise the micro code must be duplicated and the size of the micro code ROM will increase. Variable passing require additional pipeline registers between higher sequencing levels and the APU. Thus, there is a trade off between increasing the size of the micro code ROM and adding registers. The DFSM can be connected to the address ROM as well and case statements can be used to choose different memory locations when the micro code is executed.

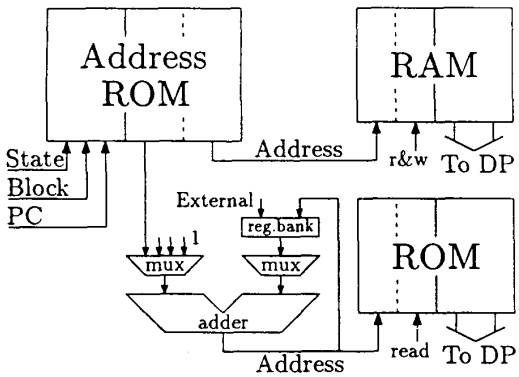

Figure 5: Memories with APU.

The APU can be implemented in two ways. Either with a single address ROM, or with an address ROM and an address processor. In applications with few memory references an address ROM without an address processor is sufficient (RAM in figure 5).

If more memory references are used the size of the address ROM will increase. The address processor implementation will then significantly reduce the size of the address ROM (ROM in figure 5). A memory address, either from the address ROM or from another module, is stored in a register and is used to compute following memory locations. The address processor is controlled by operations specified by the CUS depending on the selected implementation. Address processor operations are treated in the same way as other data path operations and can be part of case statements. Signals from the address processor, overflow signal and sign bit from the adder, are used as status signals to the DFSM. These signals can be used to control incremental loops in a micro program.

An address processor can support several memories or one address processor can be implemented for each memory. Different memories in a processor can use various strategies for the APU and different complexity of the address processor. The complexity of the address processor, number of registers and number of inputs to the multiplexers, is synthesized depending on the application.

5 Application examples

The tools have been used to design a speech scrambler chip for mobile telephones and a chip for digital adjustment of quadrature modulators.

In the scrambler the speech is split into four frequency bands which are transposed and mirrored before transmission. The algorithm requires four 6th order IIR filters at the input, followed by a down-sampler, a multiplexer and an up-sampler. The same filters are used at the output to add the different bands together, figure 6. One data path is used for all of the filters.

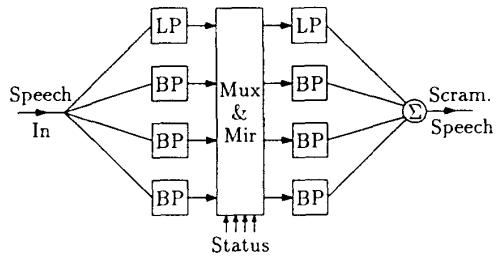

Figure 6: Principle of the speech scrambler.

Multiplexing and mirroring are controlled by external signals connected to the DFSM. The multiplexing is handled by controlling the APU to a data storage module. Depending on the state of the DFSM different data will be sent to the output filters. The chip size is 7x6 mm in a two micron technology and contains about 20 000 transistors , figure 7.

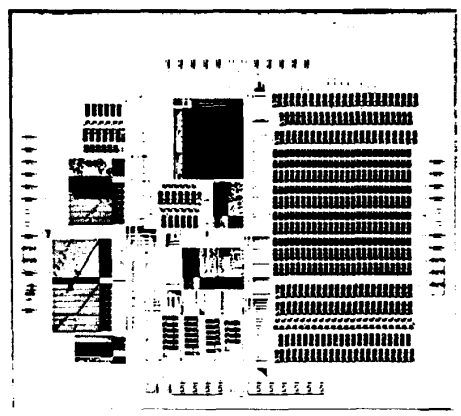

Figure 7: Die photo of speech scrambler chip.

The other application is a post-processor to a waveform generator, either a DSP or a look-up table ROM. The designed processor compensates for imbalances in Radio Frequency (RF) quadrature modulators for digital communication [5]. Traditional methods for correction of these errors usually involve improving the RF section. An alternative method applies corrections to the baseband signal, either digital or analog. The designed digital chip should be placed between the waveform generator and the digital to analog converters, figure 8.

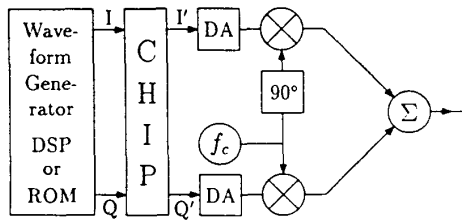

Figure 8: Correction of quadrature modulators.

The controller handles not only the sequencing of the data flow but also the communication with the host DSP. This interface communication is performed by the DFSM, control signals connected to I/O-units, and the start of sequence signal.

The algorithm requires only four micro instructions on the designed processor architecture. A corresponding implementation on a TMS320C25 processor requires more than 20 instructions for the same function. The chip size is 6x6 mm in a two micron technology and contains about 18 000 transistors , figure 9.

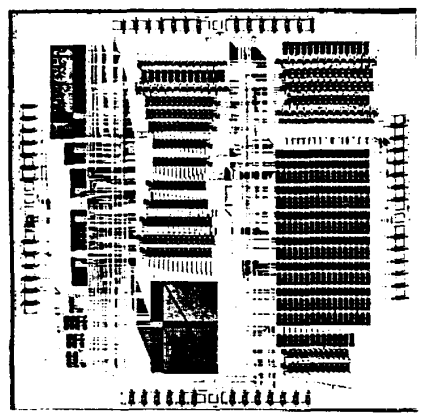

Figure 9: Die photo of correction chip.

6 Conclusions

The Control Unit Synthesizer has been developed for synthesis of controllers to arbitrary digital signal processors. A behavioral description is read for each processor and no constraints are put on the architecture.

Complexity and partitioning of the controller is dependent on the structure of the micro program. Therefore, it is easy to try different strategies to find a good solution. A complete controller and its interconnections with the data path is synthesized with specified memories and an address processing unit. A controller is synthesized with basic logic building blocks and interconnections are specified in a generated netlist. This is crucial in order to be flexible and to interface to various vendors, cell libraries, and CAD-systems.

Our applications are targeted at digital radio communication. Therefore, a hierarchical controller architecture suitable for algorithms frequently used in these applications has been developed. A finite state machine architecture as an alternative to the counter based architecture is currently under development. Work is also presently performed at adapting the Control Unit Synthesizer to a C scheduler [6] and to further investigate implementation of parallel processors on one chip.

The Control Unit Synthesizer is a part of a complete design system for application specific digital signal processors. The design of two very different applications proves the flexibility and the usefulness of the developed tools.

References

[1] Mats Torkelson. Design of application specific digital signal processors. Technical Report LUTEDX/(TETE-1004)/1-158(1990), June 1990.

[2] L. Brange and M. Torkelson. A Basic CAD-tool for module generation. In *ESSCIRC'89*, 1989.

[3] Khalid Azim. *Application of Silicon Compilation Techniques to a Robot Controller*. PhD thesis, University of California, Berkeley, Sep 1988.

[4] J. Rabaey, H. De Man, J. VanHoof, G. Goosens, and F. Catthoor. *Silicon Compilation*, chapter 8. Addison-Wesley, 1988.

[5] M. Faulkner, T. Mattsson, and W. Yates. Adaptive Linearisation Using Pre-Distortion. In *40:th IEEE Vehicular Technology Conference*, 1990.

[6] Kenneth E. Rimey. *A Compiler for Application-Specific Signal Processors*. PhD thesis, University of California, Berkeley, Sep 1989.